A multi modal interface for a visually impaired pupils educational environment (long)

Antonio Pintus, Cristian Lai
Centre for Advanced Studies, Research and Development in Sardinia, Italy
{pintux,clai}@crs4.it

Claude Moulin
UMR CNRS 6599 Heudiasyc
Université de Technologie de Compiègne, France
claude.moulin@utc.fr

Dominique Archambault
INSERM U483 / INOVA, Université Pierre et Marie Curie
France
dominique.archambault@snv.jussieu.fr

ABSTRACT (Paper ID 27)

The aim of this paper is to describe a multi modal user interface integrated within an architecture allowing visually impaired pupils and sighted people to work on the same station. This architecture is developed to face the problem of the inclusion of visually impaired pupils in mainstream education. The system is able to display the same original multi-media content converted in different ways on screen, Braille terminal and vocal synthesizers. It accepts various interactions from keyboard, Braille bar and mouse. Both blind and sighted users can manage the same tools, cooperate together on the same content, and thus perform the same activities. Blind pupils may be assisted by sighted teachers or tutors that don't know how to read Braille characters.

1. INTRODUCTION

This paper describes the elements and functionalities of a multi modal user interface (UI) integrated in a distributed architecture. This system, intended to facilitate the inclusion of visually impaired pupils in mainstream education is one part of the European Vickie (Visually Impaired Children Kit for Inclusive Education) project. This one [1] aims at giving more access to electronic documents and building a flexible and open software [8] for helping pupils to work and communicate with sighted and blind people.

The system installed on a school network provides services with pedagogical objectives and may be considered as an electronic schoolbag. One can find there various tools such as a calculator, games, a web browser or a mail service. Services can also be modules specially designed to deal with specific document such as a diary, a dictionary or an exercise book. Most of them are specially designed for visually impaired pupils and they allow to substitute actual documents or objects that other pupils can manipulate. Users are not really conscious about service existence. They work with documents or utilities ignoring the service level.

The UI addresses several input and output devices, such as screens, keyboards, mouse, Braille components, vocal synthesizer and printers. The system is able to translate the same original and multi-media content according to the particular devices: the screen shows the output in a graphical way, the Braille device in a tactile manner, the vocal synthesizer in an oral manner. For instance, pupils can choose to listen the voice instead of reading long texts. Some complementary information or explanation are only sent to text to speech engine. The system accepts keyboards, mouse, and Braille bar as normal input components. For example a teacher can use the keyboard to correct a wrong answer written from the Braille terminal.

Obviously, each service doesn't contain the UI but must call it, exchanging with it the necessary elements for describing the interface structure. Thus, services are independent pieces of software which have to follow a common model in order to be integrated in the system and to interact in the same way with the user. In the architecture the UI fulfills three objectives: be the user entry point in the system, manage the services and deal with devices.

The inclusion of visually impaired students in mainstream education has also to take into account and favor the activity of sighted people, such as teachers, tutors or parents. For this reason the UI is intended to represent the same content for all users. A typical scenario illustrates a classroom with every blind pupil sit in front of a computer equipped with the necessary devices. We may add that the screen attached to this station can be oriented towards the teacher's direction or any other direction. Generally, the other pupils don't work on computers. The activities proceed normally: all users, sighted and blind, work with the same tools either actual objects or electronic ones; teachers perform parallel activities according to the pupils' characteristics. They control either hand written pupils' work or follow on the screen the progression of others. They are supervisors of the work done in the classroom: they give advices, make corrections, etc.

In this paper we present a technical solution which allows multi modal access and interaction to content. We show how the user interface is integrated in the global architecture. We describe the model of elements exchanged between services and user interface and the way that all devices are and must be synchronized to provide a real multi-modality.

2. THE BRAILLE INTERFACE

2.1 The Braille System

The Braille system was invented less than 2 centuries ago by Louis Braille, a young blind pupil of the Royal Institute for the Blind in Paris. It is a tactile alphabet that allows to read with the fingers. It is the only tactile system that really allows a reading process, that is the same cognitive process than ordinary reading with the eyes and not a simple deciphering process. In the Braille alphabet each character is made of raised dots that can be read, after a training, exactly like reading with the eyes. In particular it can be read with a moving finger, ie the reader doesn't stop on each character.

Each character of the Braille alphabet is based on a 6 dot positions cell compound of 2 parallel columns of 3 dots, numbered 1 to 3 from top to bottom on the left column and 4 to 6 also from top to bottom on the right column. In the Braille cell, each of the 6 dot positions may be raised or not, making 64 different combinations, representing 64 different symbols. For example 'c' is represented as '14'. The 64 combinations are not enough to represent all characters, and it is needed to compound two Braille characters to obtain uppercase letters and digits.

2.2 Braille Devices

Braille devices usually provide a raw of Braille refreshable cells, function keys, and optionally cursor routing keys (for pointing) and Braille keyboards, making them input and output devices. A Braille refreshable cell is made of a framework with two columns of holes containing a mobile dot, that can be raised or lowered using a piezo electronic technology. The Braille devices use usually 8 dots instead of 6. These additional dots may be used for simulating a cursor, or for using a 8 dots Braille code avoiding to have a character made of 2 cells.

Braille devices usually have 20, 40 or 80 cells and only 1 row, based on the division of the old 80 character video display. In most models today, a small key is attached to each cell, called cursor routing key, that allows direct pointing to a character (like a mouse on a screen). A first category of Braille devices can only be used with a computer and will be used jointly with an ordinary keyboard. A second category include autonomous possibilities (note-takers) and have an embed Braille keyboard.

2.3 Braille Device Software

The market of Braille devices is made of more than 15 manufacturers, and each of them uses his own communication protocol between the device and the computer. For our project, it was necessary to access directly to Braille devices for controlling the content displayed on the cells. So, it was necessary to use a software library to easily access to Braille devices via a simple API. Libbraille [10] is a portable programming library that provides all the functions required by applications that need to drive directly Braille displays, providing all low level interactions: write a text string on the display, raise Braille dots independently, get keys that have been pressed (function keys, Braille keyboard or cursor routing keys). The library was developed using a very portable `C' code but thanks to the "Simplified Wrapper and Interface Generator" software (SWIG[1]), bindings with others languages have been generated (currently Python and Java).

[1] - http://www.swig.org

The use of Braille modality implies that all documents that will be displayed by the system provide data that fit the presentation rules of this modality, that is that all the information contained in the document are textual or have at least a textual alternative. Unlike usual cases of multimodality where the same user may use several modalities to access to the same information, the blind user has not the possibility to access the graphical information in other way than having a textual alternative.

3. THE GLOBAL SYSTEM

3.1 The Architecture

To better localize the position of the multimodal user interface in our aims, we are going to see a general view of the system distributed architecture. It is structured in two main parts: a server one and a client one. On the server side, an application server, deploys and publishes services in a service portal (they may run in the same station but it is not necessary). This essentially uses the Jini technology which allows server and portal easy localization by clients, from anywhere in a LAN. The application server uses its local file system to manage published services and users' resources. This feature performs a great role in flexibility concerning configuration and maintenance operations.

A second module of the architecture is composed of UI software running on client stations. The UI fulfills three objectives: be the user entry point in the system, manage the services performing discovery and activation steps and deal with devices. The UI discovers services in the portal, make them move to the client station and activates them locally. It also allows services to find personal resources and documents. Hence, this module is responsible for the interactions with the server part of the architecture, as shown in Figure 1.

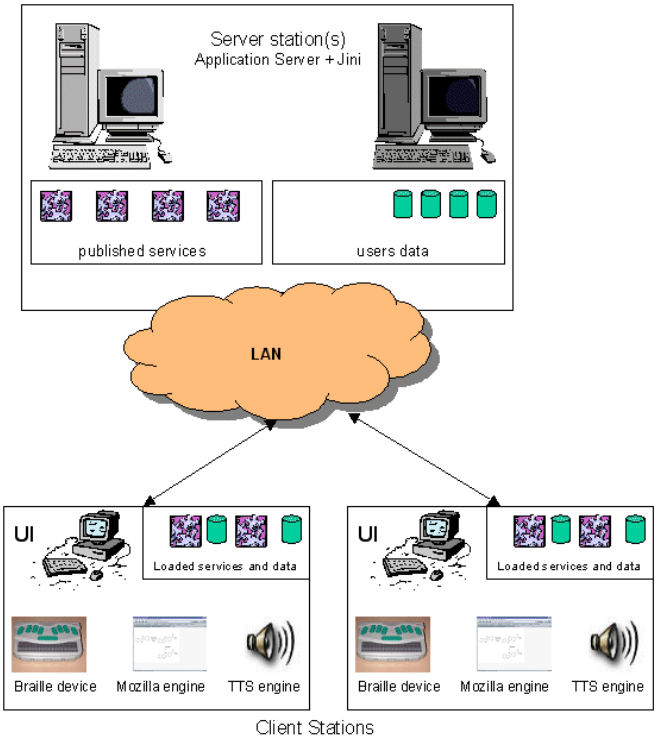

Figure 1 - System architecture.

3.2 Service Structure

Each service must use the UI to interact with the user. This imposes a common model for all the services. This is possible through the definition of programming interfaces which are claimed to be respected by services developers. At one moment, only one service is active and the user can switch from one service to another.

A service may be seen as an automaton maintaining its current state. A service changes state when receiving a user command. It then sends to the UI the material required by the interaction with the user. The service is not responsible of the state of this material during the user interaction. It is the UI role to deal with it. For example, if the user needs to perform an other service, the former completely ignores the substitution.

The multi modality is not assumed by services themselves (so services developers don't care about it): that would be a conception mistake in this kind of architecture. Services do not build themselves the user interface but they delegate this role to the UI, sending it a particular internal document. One of the UI tasks is to create the adequate user interface from the material supplied by the service for each device.

4. UI STRUCTURE

4.1 UI Elements

The UI consists mainly of two elements: the Service Interface (SI), which deals directly with services and the Device Interface (DI) which deals directly with devices. As shown in picture Figure 2, services and SI exchange information thanks to a particular internal format. The SI communicates directly with the DI, too. The SI has also the responsibility of keeping the interface state when the user switches from one service to another one without sending a request to the former.

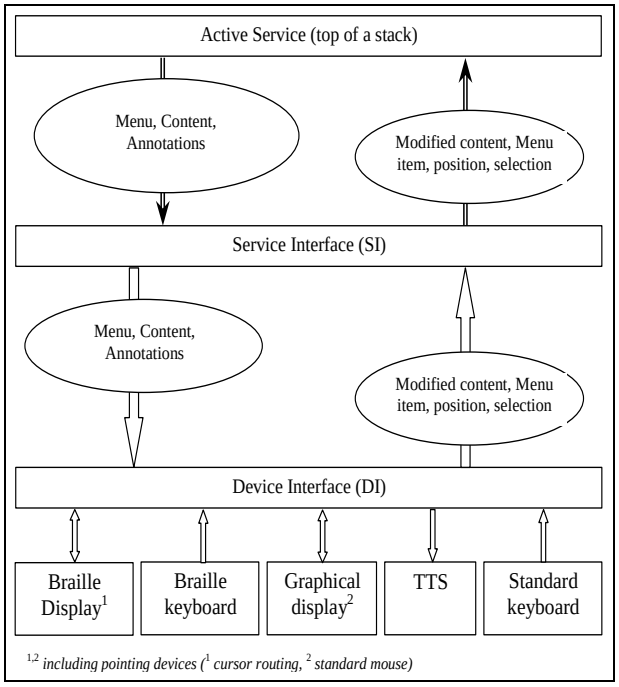

Figure 2 – UI elements and exchanged data.

4.2 Internal Format

In the 2.3 section we said that all documents that will be displayed by the system have to contain data that fit the presentation rules of the Braille modality, i.e. that all information contained in the document must be either textual or have at least a textual alternative.

Services must send objects that serves as a model for the creation of the interface and the interaction. One of these objects describes the displayed elements (text, images and sounds) and the elements that can be activated (button, checkbox, etc.). It can be saw as a XHTML [12] document and must respect an internal format. This information is exchanged between SI and DI during work sequences.

From this document, texts will be displayed line by line on Braille terminal and directly on the screen with an emphasis on the line actually displayed on the Braille cells. Images will be displayed on the screen and their legend on Braille terminal. Objects that can be activated appear on the screen and have a counterpart in Braille. The following sections describes the objects exchanged in the UI.

4.3 Information Exchanged in the UI

UI and services (thanks to SI mediation) communicate through messages and answers to these messages. The Message represents the information sent to the service to signal it that a user action is done (i.e. when a menu voice is selected). This action induces a change of service state. The Message Answer represents the service answer to a user request and is communicated to the UI. It contains all elements that a service sends to the SI in response to a message;

4.3.1 MessageAnswer

It contains mainly the document to show. It is composed of three elements:

- the content that the user interface has to display,
- a menu composed of the main UI menu and a contextual menu of the service,
- a list of annotations giving additional information on content.

4.3.2 Message

A Message consists of four elements:

- the modified document sent before by the service,
- the menu item selected by user,
- the selection made by the user in the current document,
- the cursor position.

4.3.3 Content

The content is the document that the Service sends to be displayed. It is the core of the information in the UI. It contains two elements :

- the document the Service has to manage with; it may be considered as a XHTML document. It described the displayed elements and the widgets,
- the style sheet used in the document mainly for screen display purpose.

4.3.4 Menu

The menu allows to activate a service or a UI command during the work. The menu is composed of two parts: the UI main menu, containing the general actions allowed by the environment, such as “change document”, “exit”, etc.; the other part is the service menu. Every service provides a contextual menu according to its features and state. The service provides only the selections available for the specific working sequence, such as in a typical word processor when several menu items are hidden if not necessary.

A menu can be represented by a XML file (see Figure 3). Every menu item contains several information, such as

- a text to speech string field that a vocal synthesizer may use for vocal messages
- a key string that unequivocal identify the menu item
- a shortcut string used to speed up the menu item selection by keyboard
- one or two modifiers (the second is optional) strings that represent the modifiers keyboard keys (i.e. ALT, CTRL, SHIFT) used in conjunction with the shortcut.
- a string of properties (if necessary) in the form of “key1=prop1;key2=prop2;…” useful to add no hard-coded properties to menu items.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<menu>

        <menu-item>
                <tts>menu item</tts>
                <key>a</key>
                <shortcut>ENTER</shortcut>
                <modifier1>ALT</modifier1>
                <action>select</action>
                <label>a - select a word</label>
        </menu-item>

        <menu-item>
                <tts>menu item</tts>
                <key>b</key>
                <shortcut>F1</shortcut>
                <modifier1>ALT</modifier1>
                <action>exit</action>
                <label>b - exit</label>
        </menu-item>

</menu>
```

Figure 3 - Menu file.

5. INTERFACE MODEL

5.1 Annotations

Sometimes, it is not enough to only present text but it is necessary to add some information to words or sentences[9]. For example, it may be pedagogically important to emphasize verbs in a text. That may be the work of a teacher that shows a grammar lesson, or the duty of a pupil doing an exercise. In one case, the student receives a text with emphasized words, in the other case, the student must indicate the words to emphasize. Both cases can also appear together.

We followed the concept of annotations because we need extra information on text. Generally, annotations are added by a reader according to personal objectives [6]. In our case it is true for teachers, not really for pupils. Studies show that annotations with pedagogical objectives are generally shared by a large community, even if their formulation may be different, and define a semantic system [7] that can be represented by an ontology. It is possible to find now in several researches, specifications and patterns of annotations [3].

In our case, the problem was to define the anchor and the multi-modal display of the annotation content. In Braille strings, we indicate the annotations by a special character (at the beginning and the end of the annotated portion). Activating a key, the user can listen to the content of the annotation (“emphasized text, for example”) while reading the annotated words. On the screen, this text is displayed with specific colors and font characteristics. In the other way, for annotating a piece of text, a user must first select it and activate a menu item which proposes an annotation list.

5.2 Selection and Position

An important goal of the UI is to identify where the user is pointing in the document present in the interface, and the selection context performed by the user. The message selection element provides information such as:

- the text selected within the document,
- the start and the end index of the selection.

The message position element give a reference as a XPATH, XPOINTER information relative to the associated document tree

6. INTEGRATION MODEL

As hinted in section 4.1, the DI module is responsible of the interactions with terminal devices. The integration model is essentially based on event notifying mechanisms, coming from or going towards devices. User actions, performed on connected terminal devices, are propagated towards the DI, and in real time, notification answers inducing modifications on output terminals are sent to dedicated device controllers. The respective views are coherently synchronized according to input actions.

For example, consider the use case in which a pupil is working on a text, adding characters through the Braille terminal connected to the system: the text, thanks to the system multimodality, is simultaneously updated on the screen. All current views need to be updated to make them consistent. In this way a sighted assistant or teacher can follow the pupil’s work step by step.

To achieve this behavior each device is controlled by an engine and the DI manages a main engine. Events are exchanged between controller engines and the main engine, notifying actions to trigger. A view on a device is updated when a controller engine receives such a notification.

6.1 DI Engines

We designed an extended Model View Controller organisation in order to implement the multi-modality. The Figure 4 shows the organisation of the device interface. Each device is under the control of a specific engine. The view corresponding to a device is updated by this engine. Receiving the interface model document that describes the interface objects, the main engine has to build

working objects to follow the state of the interaction with devices. For example, this state contains information on the cursor position, highlighted text, etc. This elements must be considered as the model of the main engine. When receiving an event, it prepares elements for each engine (Braille, Screen and TTS engine) and sends an event to activate them. Then, each engine can deal with its device.

When a user activate a key (or a combination of keys) the pilot engine sends an event back to the main engine[5]. This event contains a qualified action that represents the action associated to the key (LineUp, PageDown, RoutineClick, etc.). All these actions are described by tables in configuration files so that the system can be adapted to users or countries. Receiving an event the main engine interprets the qualified action and adapts the interface model working elements. Then, it gives an answer to each engine.

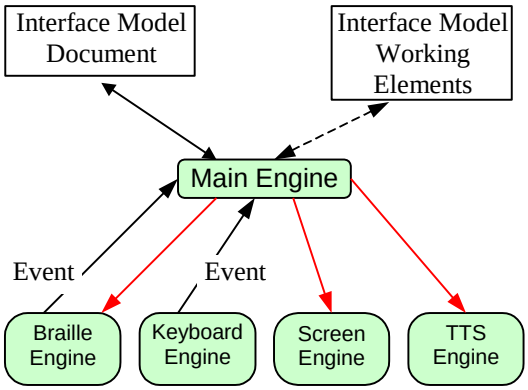

Figure 4 - Device Interface (DI) engines.

When a qualified action corresponds to a menu item activation, the main engine modifies the interface model document from the working elements and notifies the SI. If the menu item corresponds to a menu command then the SI notifies the service that responds with another interface model. If the menu item is a system command, then the SI maintains the interaction state (the modified interface model document) without notifying the service. For example, if the command is a switch towards an other service, the SI sends to the DI the interaction state of the new service.

6.2 Interaction Model

Humans use several senses simultaneously to explore and experience the environment. Technological or human limitations often prevent computer-based systems from providing genuine multimodal displays [4].

User interaction is supported by several input and output devices. What sighted user see through the screen is feel by the blind student in the same way thanks the Braille devices. The actions performed through the keyboard became the same if executed via the Braille bar. Every action mainly aims at modifying the object shown by the user interface. The transformation imply the updating of the showed element.

If the user types the 'A' key in the keyboard, means to add the character 'A' in the document. The same action will have the same effect if performed through the Braille bar typing the analogous ASCII 'A' character in Braille alphabet.

Moving along the document thanks the arrow keys is the same than type the moving button in the Braille bar. The actions becoming from the device interaction can be divided in two categories:

- to modify the content
- to arrange the communication with services

The device is responsible of the right arrangement of the interaction mechanism.

7. IMPLEMENTATION

In the development of software applications, a large amount of time is dedicated to the study and realization of the program interface [11].

Technical solution has been based on several technologies such as Java, C/C++, JNI (Java Native Interface), Mozilla, Free TTS. Vickie architecture aiming at the code mobility, the portability in several system platforms, the modularity, is developed in Java which allows to satisfy every of this characteristic. The UI, integrated in the architecture, is composed of several components. We have used the *libbraille*[2] library to connect to tactile devices. The library, developed in C language, makes possible to easily access Braille displays and terminals. Graphical display is based on Mozilla engine, an Open Source software written in C++ language, which provides several modules, useful for HTML, Math ML rendering, etc. In this way we faced the problem of mathematics for blind in front of the enormous importance it has reached in the blind community.

The integration of the various modules developed in several languages has been performed via JNI (see Figure 5), which allows to manage native code within Java applications.

The two modules compounding the UI, written entirely in Java, are the last "appendix" of the pure-Java architecture towards the devices. In reality, between the DI and terminal devices there is a further layer that consists in JNI modules which interacts with the counterpart modules written in C/C++ which drive directly the several devices, e.g. to deal with Mozilla engine or Braille devices. This JNI-C/C++ layer communicates with VDI through a listener-notifier mechanism, in particular for the views synchronization related with the actions required by clients working at terminal devices.

The DI is written in Java and it addresses to devices throws a bridge created by the adoption of JNI technology and C/C++ ad hoc libraries. JNI is a Java framework which allows calling native methods within Java code. In this way we can interact with the native modules that drive the particular device.

In particular: the DI can call native methods of Mozilla API (written in C++) for its integration, it can use the Libbraille, C libraries, for the interaction with Braille terminals and it can face the integration with TTS engines using their API libraries.

[2] http://libbraille.sourceforge.net/

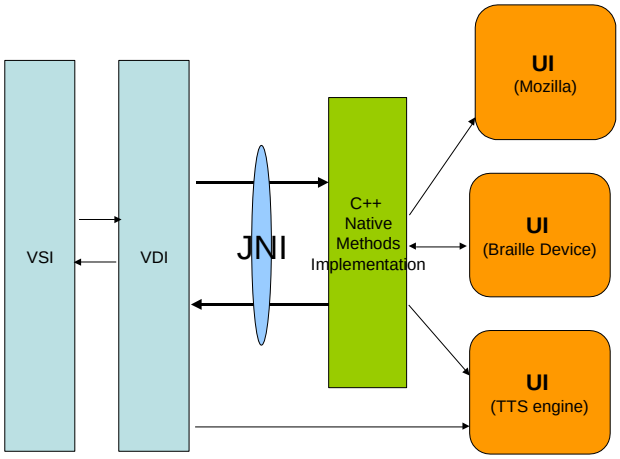

Figure 5 - Java/C++ integration.

7.1 Vocal Synthesizer

To face the voice synthesis we have chosen the Free TTS technology. It allows to describe vocally contents which are normally viewed by sighted users. It's written entirely in Java. Thanks to its characteristic we don't have to integrate it within the UI through JNI but we can directly use the FreeTTS Java API to send speech messages.

FreeTTS is a speech synthesis system based upon Flite: a small run-time speech synthesis engine developed at Carnegie Mellon University. Flite is derived from the Festival Speech Synthesis System from the University of Edinburgh and the FestVox project from Carnegie Mellon University. Currently, the distribution comes with these 3 voices: a low quality, unlimited domain, 8kHz diphone voice, called kevin , a medium quality, unlimited domain, 16kHz diphone voice, called kevin16, a high quality, limited domain, 16kHz cluster unit voice, called alan.

As recommended by its developers, we use the Java Speech API (JSAPI) to interface our system to FreeTTS. The JSAPI interface provides the best method for controlling and using FreeTTS.

Mainly, to use the engine, we have to submit the content in input. The processed output consists of the equivalent narrate text.

In future other TTS engines may be used, either written in Java or in native languages, it doesn't matter, the system is ready for an eventual integration.

7.2 Braille Devices Interfacing

To access easily Braille devices we used the "*libbraille"* API. The library is organised in a modular way. A first layer provides a simple API to developers. At initialisation and according to a configuration file, this first layer will load a device dependent module. This module implements the low level interaction with a given Braille terminal protocol and depends on the manufacturer and model of Braille terminal.

These modules use a lower level layer that provides a set of common portable functions to all drivers in order to communicate through the serial port, or to log some debug information.

7.3 Graphic Interface

As the UI internal format manages XHTML content, we decided to use an existing sighted user view. The final choice has been Mozilla, because the Open Source features, the modularity and the possibility to extend his functionalities. The best way to integrate Mozilla within an own application, in order to make an easy reuse, is to use the Embedding-Gecko SDK. The main features we used are the possibility to show and edit the content and the menu adaptation according to the MessageAnswer sent by the service. To this and other implementations Gecko provides a set of in-house programming technologies all based around an idea of object encapsulation. Some functionalities beyond basic browsing are always embedded with Gecko, some other are available purely as options. For our purpose we necessitate the support for network, CSS, DOM, XML.

8. CONCLUSION

The role of the multimodal user interface can be sawn within a flexible environment that could be considered an important tool for blind and bad sighted pupils. The system runs on the most popular platforms such as Windows and Linux. The environment has been built for the good of sighted teachers to prepare didactical material for visually impaired pupils and to give easier access to resources such as electronic books. About a dozen of primary or secondary schools in France, Ireland and Italy have been chosen to test the usability of the system and mainly its interface. Nevertheless the system manages with an own internal format, it can be considered as an open environment toward others format of documents, such as DAISY [2], ANSI/NISO, etc. The interface model presents true multi-modality and the simultaneous display on Braille terminal and screen is maybe the strongest help feature for inclusion of pupils.
Moreover, the system allows the collaboration between both blind and sighted pupils. Multi modality allows blind users to interact with the system thanks the Braille device, the keyboard and the vocal device. Sighted people use the typical equipment such as the screen, the keyboard and others analogue devices.

9. ACKNOWLEDGMENTS

The Vickie project is funded by the European Commission[3], on the IST (Information Society Technologies) Programme 2001 (FP5/IST/Systems and Services for the Citizen/Persons with special needs), under the reference IST-2001-32678.

10. REFERENCES

[1] Archambault D. & Burger D., The Vickie Project, Visually Impaired Children Kit for Inclusive Education, 8th International Conference on Computers Helping People with Special Needs (ICCHP'2002), p. 90-97, Linz, July 2002.

[2] DAISY Consortium, http://www.daisy.org, 2002

[3] Desmoulins C., Mille D., Pattern-based Annotations on E-books : From Personal to Shared Didactic Content, International Workshop on Wireless and Mobile Technologies in Education, Växjö, Sweden, august 2002.

[3] The content of this paper is the sole responsibility of the authors and in no way represents the views of the European Commission or its services.

[4] Fontana F., Rocchesso D. & Ottaviani L. A structural approach to distance rendering in personal auditory, ICMI'02 conference, p. 33-38, Pittsuburg, October 2002.

[5] Kleindienst J., Seredi L., Kapanen P., Bergman J. CATCH-2004 Multi-Modal Browser: Overview Description with Usability Analysis, ICMI'02 conference, p. 442-447, Pittsuburg, October 2002.

[6] Marshall C., Annotations: from paper books to the digital library, ACM Digital Libraries '97, ACM press, Philadelphia, PA, p. 131-140, 1997.

[7] Marshall C., Toward an ecology of hypertext annotation, ACM Hypertext'98, ACM press, Pittsburgh, PA, p. 40-49, 1998.

[8] Moulin C., Giroux S., Archambault D., Carboni D. & Burger D., A distributed document oriented architecture for rendering services to visually impaired students, 8th International Conference on Computers Helping People with Special Needs (ICCHP'2002), p. 329-336, Linz, July 2002.

[9] Petrie H., Fisher W., Langer I., Weber G., Gladstone K., Rundle C., Pyfers L., Universal Interfaces to Multimedia Documents, ICMI 2002 conference, p. 319-324, Pittsuburg October 2002.

[10] Sablé S. & Archambault D., Libbraille: A Portable Library to Easily Access Braille Displays, Proceedings of ICCHP 2002 (8th International Conference on Computers Helping People with Special Needs), Linz, Austria, July 2002. — Springer LNCS 2398, K. Miesenberger, J. Klaus, W. Zagler (Eds.). — pp. 345-352.

[11] Taddei L., Costantini E. & Alon Lavie, The Nespole! Multimodal Interface for Cross-lingual Communication, Experience and lessons learned, ICMI'02 conference, p. xx-xx, Pittsuburg, October 2002.

[12] W3C, reference for XML, XHTML, XPATH and MathML, http://www.w3.org/, 2001.